TASK 3:

```
import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import davies_bouldin_score, silhouette_score

import matplotlib.pyplot as plt


# Load datasets

customers = pd.read_csv('/content/drive/MyDrive/Customers.csv')

transactions = pd.read_csv('/content/drive/MyDrive/Transactions.csv')


# --- Data Preparation ---


# 1. Merge datasets

customer_data = pd.merge(customers, transactions, on='CustomerID')


# 2. Feature engineering and selection

# a. Total spending per customer

customer_spending =
customer_data.groupby('CustomerID')['TotalValue'].sum().reset_index()

customer_spending.rename(columns={'TotalValue': 'TotalSpending'}, inplace=True)


# b. Average transaction value per customer

customer_avg_transaction =
customer_data.groupby('CustomerID')['TotalValue'].mean().reset_index()

customer_avg_transaction.rename(columns={'TotalValue': 'AvgTransactionValue'},
inplace=True)
```

```
# c. Purchase frequency per customer
customer_frequency =
customer_data.groupby('CustomerID')['TransactionID'].count().reset_index()
customer_frequency.rename(columns={'TransactionID': 'PurchaseFrequency'}, inplace=True)

# d. Merge features with customer data
customer_features = pd.merge(customers, customer_spending, on='CustomerID')
customer_features = pd.merge(customer_features, customer_avg_transaction,
on='CustomerID')
customer_features = pd.merge(customer_features, customer_frequency, on='CustomerID')

# 3. One-hot encoding for categorical features
customer_features = pd.get_dummies(customer_features, columns=['Region'],
drop_first=True)

# 4. Feature scaling
features_to_scale = ['TotalSpending', 'AvgTransactionValue', 'PurchaseFrequency']
scaler = StandardScaler()
customer_features[features_to_scale] =
scaler.fit_transform(customer_features[features_to_scale])

# --- Clustering ---

# 1. Determine optimal number of clusters (using elbow method)
wcss = []
for i in range(2, 11):
   kmeans = KMeans(n_clusters=i, random_state=42)
   kmeans.fit(customer_features[features_to_scale])
   wcss.append(kmeans.inertia_)
```

```
plt.plot(range(2, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()

# 2. Apply KMeans clustering
n_clusters = 3  # Replace with optimal number from elbow method
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
customer_features['Cluster'] = kmeans.fit_predict(customer_features[features_to_scale])

# --- Evaluation ---

# 1. DB Index
db_index = davies_bouldin_score(customer_features[features_to_scale],
customer_features['Cluster'])
print(f"DB Index: {db_index}")

# 2. Silhouette Score
silhouette = silhouette_score(customer_features[features_to_scale],
customer_features['Cluster'])
print(f"Silhouette Score: {silhouette}")

# --- Visualization ---

# Scatter plot (example)
plt.figure(figsize=(8, 6))
```

```python
plt.scatter(customer_features['TotalSpending'], customer_features['PurchaseFrequency'], c=customer_features['Cluster'])

plt.title('Customer Segmentation')

plt.xlabel('Total Spending')

plt.ylabel('Purchase Frequency')

plt.show()


# --- Report ---


# Number of clusters: n_clusters

# DB Index: db_index

# Silhouette Score: silhouette

# Visualization: Scatter plot (and other relevant plots)
```